ESIREM
ÉCOLE SUPÉRIEURE D'INGÉNIEURS
NUMÉRIQUE ET MATÉRIAUX

École associée Polytech

Fundamentals of C++.

From beginner to beyond.

oct. 2021

Lecture #06

Templates

http://ginhac.com/teaching/ITC313/2021-2022/06-templates.pdf

Photo by Nick Tiemeyer on Unsplash

Introduce how to implement Generic programming using templates in C++.

Enjoy! ☺

Today

Lecture #00
Course Introduction

Lecture #01
User-defined Data Types

Lecture #02
Inheritance

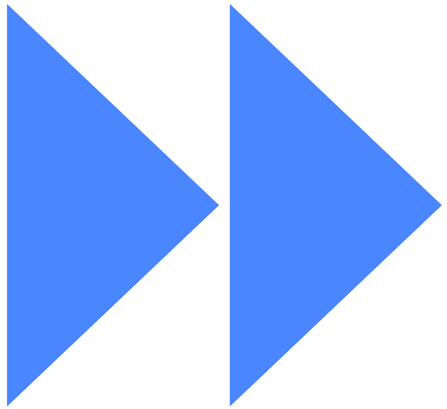
Lecture #03
Polymorphism

Lecture #04
STL Containers

Lecture #05
Indirection

Lecture #06
Templates

Lecture #07
Exceptions



AGENDA

01 – An introduction to Generic Programming

02 – Templates in functions

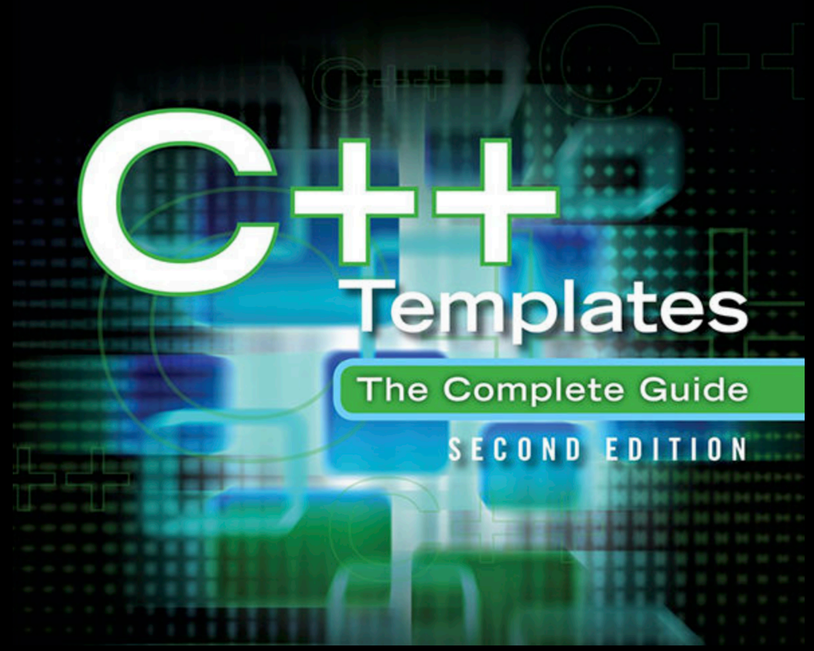
03 – Templates in classes

What is “generic programming”?

Programming often needs for creating functions or classes which perform the same operations but work with different data types.

C++ provides templates to create single generic functions/classes instead of many functions/ classes which can work with different data type.

What are “templates”?



Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type.

A template is a blueprint for creating a generic class or a function. It is a mechanism that allows a programmer to use types as parameters for a class or a function.

The compiler then generates a specific class or function when we later provide specific types as arguments.

Template in C++

STL

User-defined functions

User-defined classes

Template in C++

STL

Already learnt with array, vector, ...

See http://www.cplusplus.com/reference/stl/

and lecture 04-stl.pdf

for details

User-defined functions

User-defined classes

AGENDA

01 – An introduction to Generic Programming

02 – Templates in functions

03 – Templates in classes

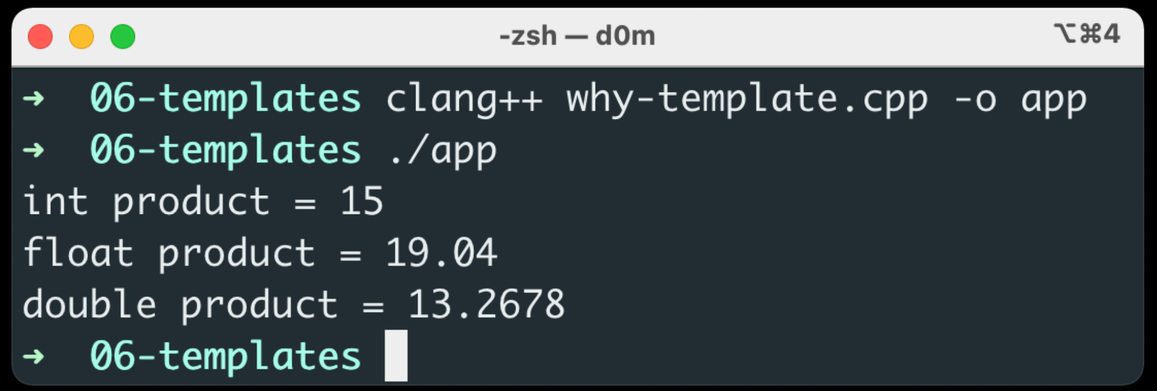
Why Template function?

why-template.cpp

```cpp
#include<iostream>

int product(int num1, int num2) {
    return num1*num2;
}
float product(float num1, float num2) {
    return num1*num2;
}
double product(double num1, double num2) {
   return num1*num2;
}

int main() {
    int i1=3, i2=5; float f1=3.4, f2=5.6; double d1=5.67, d2=2.34;
    std::cout << "int product = " << product(i1,i2) << std::endl;
    std::cout << "float product = " << product(f1,f2) << std::endl;
    std::cout << "double product = " << product(d1,d2) << std::endl;
    return 0;
}
```



Don’t repeat yourself !

Template function

In C++, the format for declaring function templates with type parameters is:

```
template <class identifier> function_declaration;

template <typename identifier> function_declaration;
```

The only difference between both prototypes is the use of either the keyword class or the keyword typename.

Its use is indistinct, since both expressions have exactly the same meaning and behave exactly the same way.

The “templated version” of product function

```
template <typename T>
T product(T data1, T data2) {
    return data1 * data2;
}

int main() {
    int i1=3, i2=5;
    int i3=product(i1,i2);
    float f1=3.4, f2=5.6;
    float f3=product(f1,f2);
    double d1=5.67, d2=2.34;
    double d3=product(d1,d2);
    return 0;
}
```

T is the templated type

Compiler internally generates and adds the code of the three functions

```
int product(int d1, int d2) {
    return d1*d2;
}
```

```
float product(float d1, float d2) {
    return d1*d2;
}
```

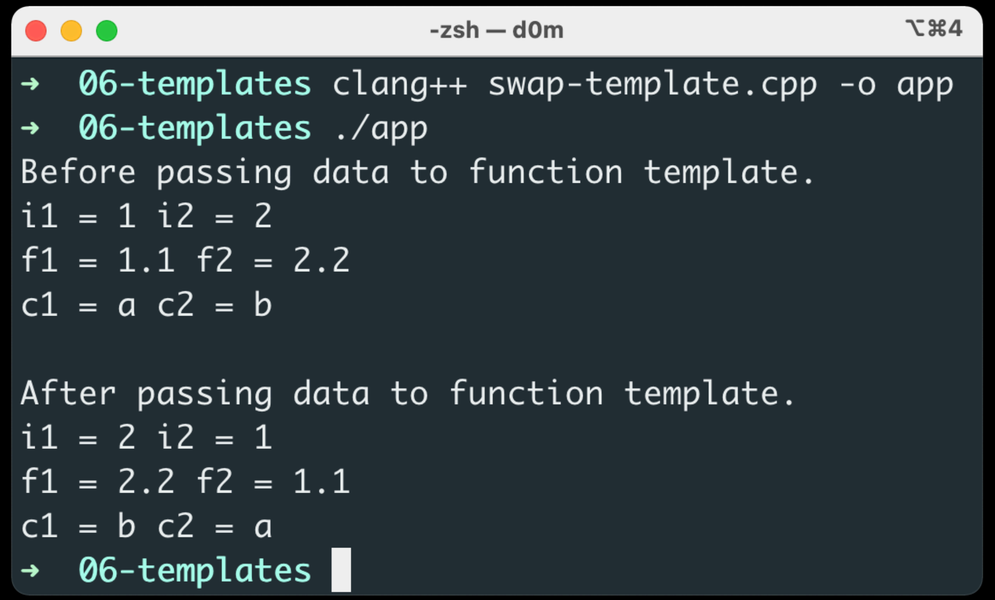
```
double product(double d1, double d2) {
    return d1*d2;
}
```

Swap Data Using Function Templates

swap-template.cpp

```cpp
#include<iostream>

template <typename T>
void swap(T& n1, T& n2) {
    T temp;
    temp = n1;
    n1 = n2;
    n2 = temp;
}
int main() {
    int i1=1, i2=2;    float f1=1.1, f2=2.2;    char c1='a', c2='b';
    std::cout << "Before passing data to function template.\n";
    std::cout << "i1 = " << i1 << " i2 = " << i2 << std::endl;
    std::cout << "f1 = " << f1 << " f2 = " << f2 << std::endl;
    std::cout << "c1 = " << c1 << " c2 = " << c2 << std::endl;
    swap(i1, i2);    swap(f1, f2);    swap(c1, c2);
    std::cout << "After passing data to function template.\n";
    std::cout << "i1 = " << i1 << " i2 = " << i2 << std::endl;
    std::cout << "f1 = " << f1 << " f2 = " << f2 << std::endl;
    std::cout << "c1 = " << c1 << " c2 = " << c2 << std::endl;
    return 0;
}
```



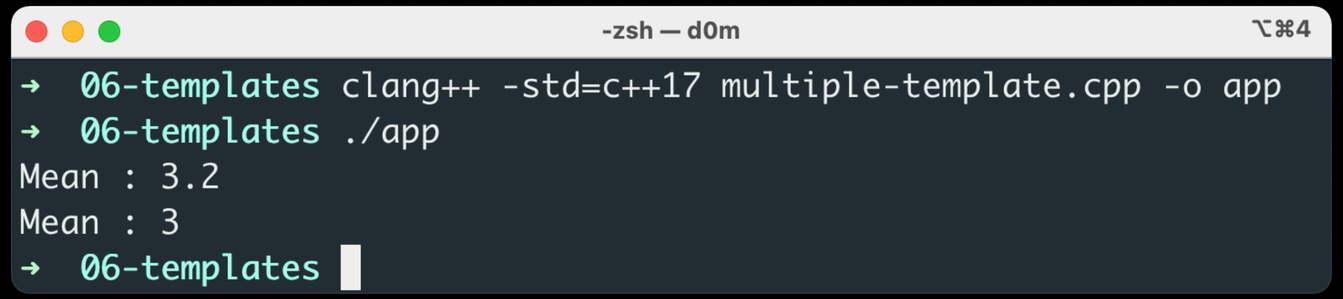
Swap function uses 2 template variables of the same type.

Using multiple types

multiple-template.cpp

```cpp
#include <iostream>

template<typename T, typename S> S mean(const std::vector<T> data) {
    S sum(0);
    for(auto d: data)
      sum += d;
      if (data.size()>0)
         return sum/data.size();
      else
        return -1;
}
int main() {
    std::vector<int> tab = {3, 5, 5, 2, 1};
    std::cout << "Mean double : " << mean<int,double>(tab) << std::endl;
    std::cout << "Mean int : " << mean<int,int>(tab) << std::endl;
    return 0;
}
```



Using default values

default-parameter-template.cpp

```
#include <iostream>

template <typename T=double, int count=3> T multIt(T x){
    for(int ii=0; ii<count; ii++) {
        x = x * x;
    }
    return x;
};
int main() {
   int i1 = 2; double d1 = 2.1;
   std::cout << i1 << ": " << multIt<int,6>(i1) << std::endl;
   std::cout << i1 << ": " << multIt<int>(i1) << std::endl;
   std::cout << d1 << ": " << multIt<double,2>(d1) << std::endl;
   std::cout << d1 << ": " << multIt<double>(d1) << std::endl;
   std::cout << d1 << ": " << multIt<>(d1) << std::endl;
   return 0;
}
```

Questions

AGENDA

01 – An introduction to Generic Programming

02 – Templates in functions

03 – Templates in classes

Template function

Just as function templates, we can also define class templates.

The general form of a generic class declaration is

```
template <class type> class classname {
    .
    .
    .
}
```

Like function templates, class templates are useful when a class defines something that is independent of the data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, ...

The stack example

stack-template.cpp

```
#include <iostream>
#include <vector>
#include <string>
#include <stdexcept>

template <class T>
  class Stack {
   private:
      std::vector<T> elems;     // elements
   public:
      void push(T const&);      // push element
      void pop();               // pop element
      T top() const;            // return top element
      bool empty() const {      // return true if empty.
         return elems.empty();
      }
};
```

Class Stack<> uses a vector of <T> elements

It implements generic methods to push and pop the elements from the stack

The stack example

stack-template.cpp

```
template <class T>
void Stack<T>::push (T const& elem) {
    // append copy of passed element
    elems.push_back(elem);
}

template <class T>
void Stack<T>::pop () {
   if (elems.empty()) {
      throw std::out_of_range("Stack<>::pop(): empty stack");
   }
   // remove last element
   elems.pop_back();
}

template <class T>
T Stack<T>::top () const {
   if (elems.empty()) {
      throw std::out_of_range("Stack<>::top(): empty stack");
   }
   // return copy of last element
   return elems.back();
}
```

Class stack<> implements generic methods to push and pop the elements from the stack

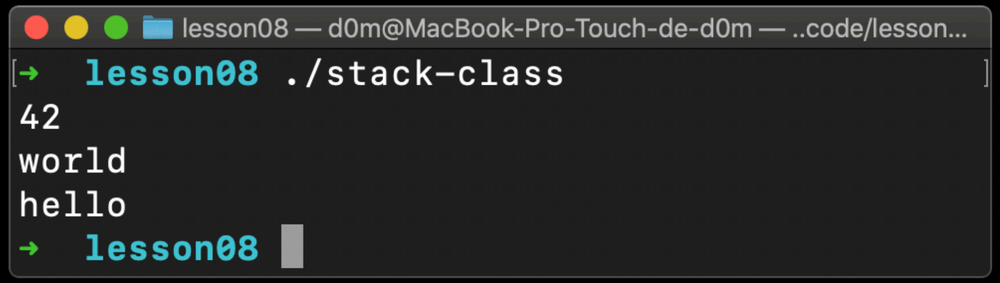
The stack example

stack-template.cpp

```
int main() {
        Stack<int>          intStack;  // stack of ints
        Stack<std::string> stringStack;    // stack of strings

        // manipulate int stack
        intStack.push(42);
        std::cout << intStack.top() << std::endl;

        // manipulate string stack
        stringStack.push("hello");
        stringStack.push("world");
        std::cout << stringStack.top() << std::endl;
        stringStack.pop();
        std::cout << stringStack.top() << std::endl;
        //stringStack.pop();
}
```



Questions

#08

Take Home Message

Templates are one of the core concepts of generic programming in OOP

Templates functions and templates classes are used to provide facility to the programmer, to write generic code that is usable in many cases.

Templates reduce the effort on coding and debugging for different data types to a single set of code

Photo by Alexander Andrews on Unsplash

Contacts

Pr. Dominique Ginhac

dginhac@u-bourgogne.fr

Come visit us at

https://github.com/dginhac/esirem-itc313